Automated Maze Generation for Ms. Pac-Man Using Genetic Algorithms

Aykut Burak Safak, Erkan Bostanci and Ali Emre Soylucicek

***Abstract*—Ms. Pac-Man has been a very popular arcade game since its release in 1982. The original game is based on a single maze structure which can make game play lose its attractiveness eventually. This paper aims to generate sets of various mazes through an evolutionary approach. A genetic algorithm was designed to create optimal mazes by specifying a fitness function to create different mazes which can allow the game to be finished by the player. Evolutionary approach was chosen due to its unique yet acceptable results. Results show that different maze structures are possible to obtain in addition to the classical design.**

***Index Terms*—Genetic algorithm, evolutionary maze generation, Ms. Pac-Man, artificial intelligence.**

I. Introduction

Automated generation of game levels is an interesting and active research field [1-4] as well as finding solutions to mazes using various techniques [5]. Such mazes allows generation of different unpredictable game environments for arcade games like Ms. Pac-Man.

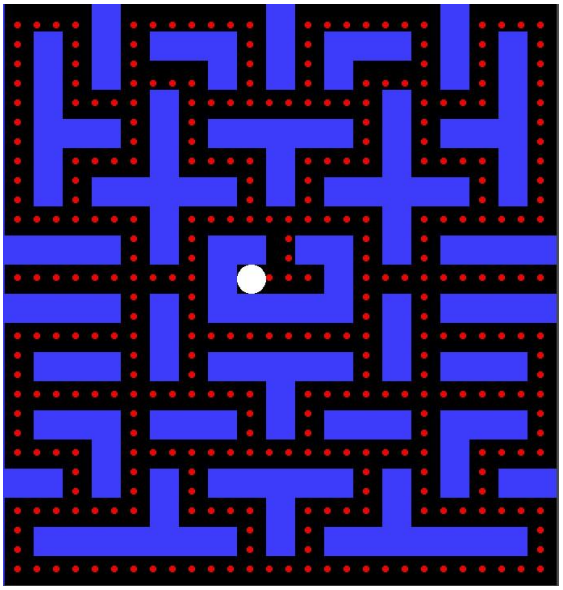
Heuristic approaches such as genetic algorithms allow finding optimal maze structures through a number of iterations in which best candidates are chosen. Genetic algorithms, proposed by Alan Turing in 1950 by the idea of a learning machine which would parallel the principles of evolution [6], are heuristic search algorithms that mimics the natural process of evolution. These algorithms are mostly used to generate solutions to optimization and search problems. Although Turing was the one who proposed the idea of evolutionary machines, Nils Aall Baricelli simulated the idea on a computer in 1954 [7]. Alex Fraser, Australian Geneticist, published a series of papers on simulation of artificial selection of organisms using genetic algorithms[8]. Hans-Joachim Bremermann published a series of papers in the 60s that adopted a population solution to optimization problems, undergoing recombination, mutation and selection.

Although Baricelli designed a genetic algorithm that plays a simple game [9], artificial evolution became a widely recognized optimization method as a result of the work of Ingo Rechenberg and Hans-Paul Schwefel in the 60s Rechenberg's group was able to solve complex engineering problems using genetic algorithms [10].

Related to Ms. Pac-Man game, there are already different studies that try to improve the intelligence of the enemies and the main character such as evolving location evaluator [11] or using tree search methods for safe locations [12]. More examples of adding the game with more sophisticated intelligence can be found in [13-17] including creation of map models or generation of an agent using Ant Colony Optimization.

This paper explores the use of genetic algorithms for automated generation of mazes for this game. Original maze of the Pac-Man can be seen at the Maze.1. The main aim here is to generate mazes that allow the game to be completed. These different mazes can be played as sequential levels for the original game which consisted of a single level.

The rest of the paper is structured as follows: Section II describes the approach for modelling this problem in order to find a solution using genetic algorithms. Section III presents the graphics library used in the work. Section IV results of automated maze generation followed by Section V where the paper is concluded.



Maze.1. Original Maze of the Pac-Man Game.

II. Approach

Genetic algorithms mimic the process of natural selection, reproduction of fit members of a population and evolution. The following sections describe the design stages for the maze generation.

A. Steps of Genetic Algorithms

The algorithm used here can be described in a few simple steps:

1) Producing new members by crossover
2) Calculating the fitness of every member in the population

Manuscript received January 22, 2016; revised July 1, 2016.

The authors are with SAAT Lab in Computer Engineering Department, Ankara University, Turkey (e-mail: a.buraksafak@gmail.com, ebostanci@ankara.edu.tr, alisoylucicek@gmail.com).

3) Mutating some random members

Describing the genes correctly and efficiently is one of the most important steps of the genetic algorithms since the structure is the piece of information that will evaluate to a solution to the problem.

B. Genes

Genes are the symbolization of an individual's solution to the problem, so describing them correctly is important for finding an optimal solution.

Genes that are used in the project have the main information about the blocks constructing the maze. This main information is the position of a block in a two dimensional space, whether the block is horizontal or vertical and it's size. Every maze in the population has 48 blocks in it and every block has four parameters, which constructs the gene for every member as a two dimensional array which has 48x4 members.

Storing genes in an array makes it easier to crossover the genes and to make necessary calculations. These mazes are then can be easily rendered by the graphics system that is used in the game.

C. Fitness of Individuals

Calculating the fitness is the most important step and a very crucial to the algorithm. This step describes what the algorithm is looking for and what will be solution to the problem, so it is really important to describe and implement this function correctly.

The fitness function used in the project was using the information about mazes; whether the maze is playable or not, whether the blocks have spread to the maze homogeneously or haven't, total count of blocks (considering the area of them), ratio of horizontal and vertical blocks.

First algorithm to determine if the maze is playable was to auto-play the game until it is finished, but this approach took too much time and was not reliable so the algorithm was reconstructed and redesigned to a much more effective and swift one. The algorithm used for this operation is as follows:

1) Select one of the dots from the maze and push it to a stack and flag it as checked
2) Pop a dot from the stack and check whether it have any neighbours (top, left, bottom, right)
3) Push any unchecked neighbours of the dot to the stack
4) Iterate second and third steps until there is no members in the stack
5) If the iteration of second and third steps are equal to the count of the dots than it means the game is playable, else is not playable.

Each fitness qualifications described above has different weight and effect on solution. The fitness of each member is calculated using the following:

```
0.4 * is_finishable - 0.1 *
intersected_block_ratio + 0.2 *
homogeneity_factor + 0.2
*  horizontal_vertical_ratio + 0.2 *
block_size_ratio
```

Each member on the formula is calculated as follows:

```
is_finishable = 1 if the maze is finishable
else 0,

intersected_block_ratio =
intersected_square_count /
total_square_count,

horizontal_vertical_ratio =
horizontal_square_number /
vertical_square_number

block_size_ratio = (square_count - 150) /
150,
```

Fitness function was designed to calculate directly from the genes but this approach had some memory issues and was hard to calculate and read. These problems have been overcome by creating a maze map from the genes as given in Fig. 1. This binary structure of the maze can easily be modelled as a binary chromosome when read in row major order.

Fig. 1. Maze map made out from genes.

This map was designed to use 1's for blocks and 0's for the dot positions. It can be considered as a 20x20 tiled map of the real maze. This approach made the calculations much easier.

D. Cross-over

Crossover, in the genetic algorithms, is used for creating new candidate solutions from the existing ones. It is a simulation of natural crossover as the genetic algorithms are the simulation of nature.

Crossover method used here is three point crossover method. Each selected candidate gene divided into four equal pieces to generate two new candidates as depicted in Fig. 2.

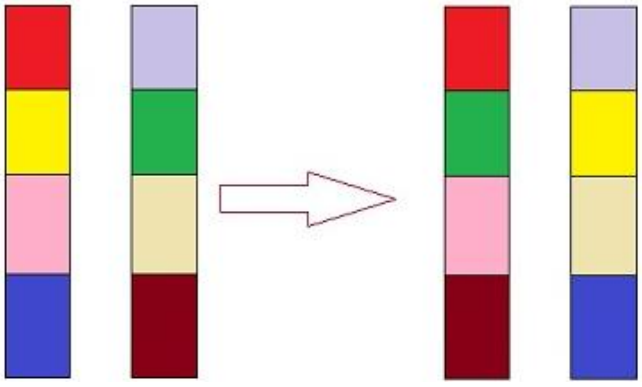


Fig. 2. Three point crossover of two candidates.

E. Selection

Supervised randomizing is used to select parent candidates to crossover. Supervised randomness is accomplished by selecting randomly while giving fit candidates more chance than the others in the reproduction process. The main idea behind this is that expectation that fit parents are more likely to produce fitter siblings for the following generations.

Roulette selection algorithm is used to apply this idea to the paper. Working principle of this algorithm is to select a random point from a wheel which was created considering the fitness of the candidates (Fig. 3). Here the likelihood of selection for reproduction of a fitter parent is higher than a parent with a lower fitness value.

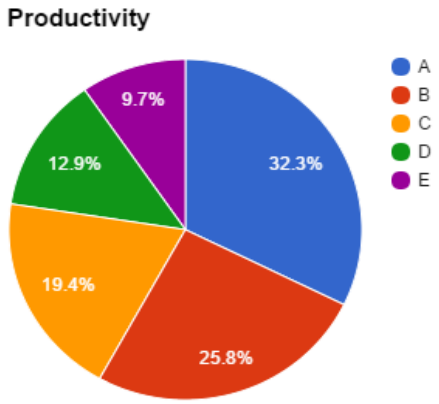


Fig. 3. Roulette selection for five candidates with different fitness values.

F. Mutations

Little piece of the candidates is mutated to create more diverse populations in genetic algorithms. The mutation ratio was selected as 0.05 and applied on random candidates' random gene parts.

III. SFML

SFML (Simple Fast Multimedia Library) is a simple to understand yet efficient to use multimedia library mainly focused on two dimensional basic graphical programmes. SFML is a cross-platform development library designed for the simple multimedia applications. It's written in C++ but have bindings for many other programming languages including Java, C, Python, Ruby, Rust, .Net etc.

SFML is an easy to implement multimedia library and easy to use. These specifications made SFML a perfect candidate for the work.

Considering the aim of the work the graphic library used doesn't have to be a strong and complicated library but simple, efficient one. SFML met all these requirements.

IV. Results

The genetic algorithm for creating different maze structure have been run many times for tuning the fitness criteria to create mazes that can actually be completed by the player. This was required since some of the maze produce by the algorithm included unreachable parts which would not allow the game to be finished. The details will be given in the following.

Elimination of the unfit candidates has a major effect on the solution because it increases the chance of the fit candidates while decreasing the number of unfit values. The results for 2000 generations are given in Fig. 4 for the case where elimination was not employed. Note that the maximum average fitness can go only up to 0.5.

Fig. 4. Average fitness of 2000 generations without elimination factor.

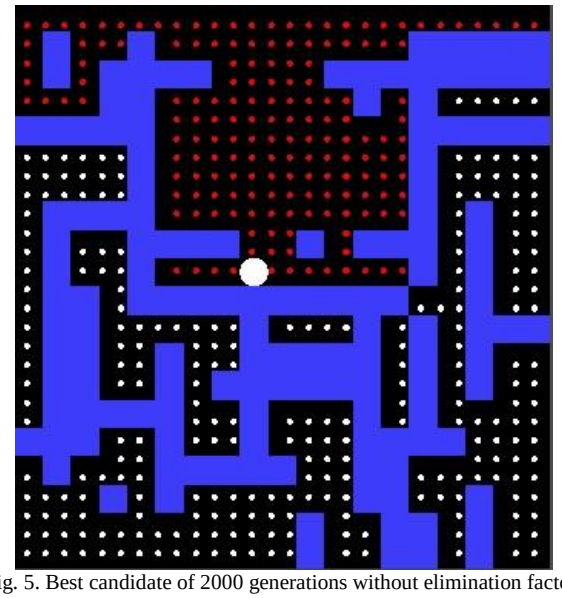


Fig. 5. Best candidate of 2000 generations without elimination factor.

The result of these low fitness values have a reflection in the generated mazes as well as shown in Fig. 5. Red dots here represent reachable pellets while the white ones represent the unreachable pellets. Here, it would not be possible to finish the game since the player cannot reach all the pellets.

Fig. 6 shows the average fitness value for 2000 generations when the algorithm was run on an initial set of randomly generated mazes. The nature of the algorithm follows an elitist approach which keeps the best-so-far individuals in the population so that these fit individuals would not be altered by any mutations. This allowed the overall fitness to go stagnant in some cases but never fall down the best results.

Looking at Fig. 7, one can see the effect of elimination factor as the reachable dot positions. Every pellet can be reachable so this maze can be finished. One problem here is that there can be corridors with width greater than one, a problem we are working on to solve.

It is important to reiterate that the average fitness for 2000 generations can reach to 0.6 when the initial fitness values were starting with 0.16. When this was not included, the average fitness could only reach values around 0.25.

Results have demonstrated that genetic algorithms give

more reliable and correct result each iteration of generation. Even though more iterations means more reliable results, randomness of genetic algorithms makes every solution unique.

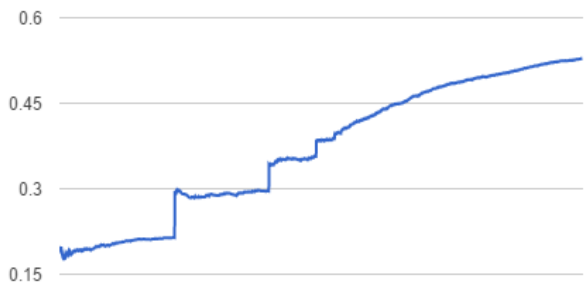


Fig. 6. Average fitness of 2000 generations with elimination factor.

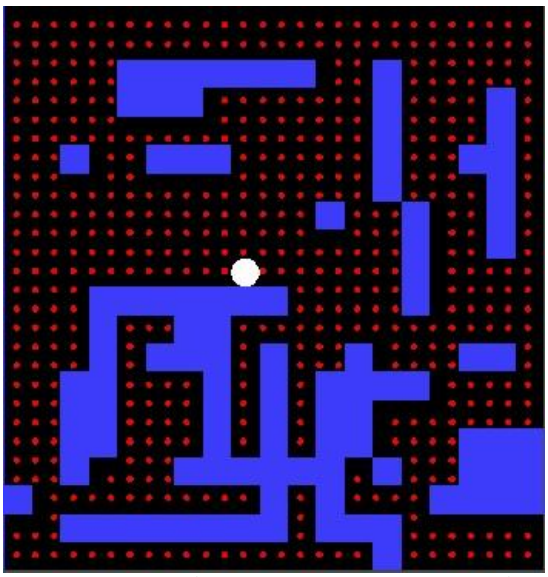


Fig. 7. Best candidate of 2000 generations with elimination factor.

V. Conclusion

Genetic algorithms are heuristic search algorithms for finding optimal solutions to the problems will ill-definitions questions by scientists. Inspired from the natural law of survival of the fittest, optimal solutions to such problems can be found through an incremental optimization process. This process has a significant amount of space for randomness which is used to avoid local optima.

Automated maze generation is also a problem that has many different difficult parameters to satisfy. This paper presented a genetic algorithm design to tackle this problem. We presented a gene design for this problem and introduced parameters to consider when the fitness functions will be computed.

Self maze constructing can give the user unique game environment every time. Uniqueness of every solution produced by the genetic algorithms made them perfect for the problem. This feature can make the game more challenging since this forces the user to change the game strategy at each different level. The ultimate aim is to enhance game play and make the game more interesting.

It was shown that heuristic algorithms such as evolutionary computation can make use of a number of criteria to generate mazes in order to improve game level designs. With a clear and well-designed fitness function it is possible to obtain much better maze design in an autonomous fashion without requiring human design once the function is defined. It is also important to note the role of randomness in such algorithms. Avoiding local optima being one benefit, the second benefit is the ability to create unpredictable sequences of genes yielding very unlikely maze designs.

Future work will investigate improving the fitness function employed here to create better maze structures. An approach for adding a more sophisticated artificial intelligence to the enemies are in our current research agenda. Adding a factor of randomness can make enemies less predictable and make the game more challenging.

References

[1] D. Ashlock, C. Lee, and C. McGuinness, "Search-based procedural generation of maze-like levels," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no.3 , pp. 260-262, Sep. 2011.
[2] Y. Okamoto and R. Uehara, "How to make a picturesque maze," CCCG, pp. 137-140, 2009.
[3] M. Foltin, "Automated maze generation and human interaction," M.S. thesis, Masaryk University Faculty of Informatics, 2011.
[4] J. Xu and C. S. Kaplan, "Image-guided maze construction," *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3, pp. 29, August, 2007.
[5] A. M. Reynolds, "Maze-solving by chemotaxis," *Physical Review E*, vol. 81, no. 6, 2010.
[6] A. M. Turing, "Computing machinery and intelligence," *Mind LIX*, vol. 238, pp. 433–460.
[7] N. A. Barricelli, "Esempi numerici di processi di evoluzione," *Methodos*, pp. 45–68, 1954.
[8] A. Fraser and D. Burnell, *Computer Models in Genetics*, New York: McGraw-Hill, 1970.
[9] N. A. Barricelli, "Numerical testing of evolution theories. Part II. Preliminary tests of performance, symbiogenesis and terrestrial life," *Acta Biotheoretica*, vol. 16, pp. 99-126, 1963.
[10] I. Rechenberg, *Evolutionsstrategies*, Stuttgart: Holzmann-Froboog, 1973.
[11] S. M. Lucas, "Evolving a neural network location evaluator to play Ms. Pac-Man," CIG, 2005.
[12] D. Robles and S. M. Lucas, "A simple tree search method for playing Ms. Pac-Man," presented at IEEE Symposium on IEEE Computational Intelligence and Games, 2009.
[13] J. DeNero and D. Klein, "Teaching introductory artificial intelligence with pac-man," presented at the Symposium on Educational Advances in Artificial Intelligence, 2010.
[14] N. Wirth and M. Gallagher, "An influence map model for playing Ms. Pac-Man," presented at IEEE Symposium on Computational Intelligence and Games, 2008.
[15] M. Emilio, "Pac-mAnt: Optimization based on ant colonies applied to developing an agent for Ms. Pac-Man," presented at IEEE Symposium on Computational Intelligence and Games, 2010.
[16] M. Gallagher, and M. Ledwich, "Evolving pac-man players: Can we learn from raw input?" presented at IEEE Symposium on Computational Intelligence and Games, 2007.
[17] P. Rohlfshagen and S. M. Lucas, "Ms pac-man versus ghost team cec 2011 competition," presented at IEEE Congress on Evolutionary Computation, 2011.

Aykut Burak Safak is a senior student in Ankara University Computer Science Department. He graduated from Bor Akin Gonen Anatolian High School. He completed two summer internships at TaleWorlds Entertainment. His work interests include computer graphics, artificial intelligence, genetic algorithms and fuzzy logic. He has been working in TaleWorlds Entertainment as a junior developer since 2014.

He is focused on developing artificial intelligence in games. His studies also focus on the graphics programming in games. He created several games using Unity 3D and SFML and a simple game engine using Open-GL for the purpose of self-development yet haven't published

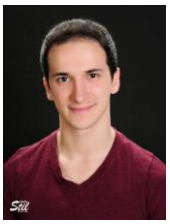
any of them.

Erkan Bostanci received a BSc degree in Computer Engineering Department from Ankara University, Turkey in 2007. Consequently, he joined the same department as a Research Assistant and completed his MSc on real-time battlefield simulation in 2009. He obtained his PhD from School of Computer Science and Electronic Engineering, University of Essex, United Kingdom in 2014 with his thesis on real-time user tracking for augmented reality.

He started working with the Gendarmarie Schools Command as a Planning Officer Designate in June, 2014 where he conducted the research for developing a vision-based system for analysing crime scenes. He has been promoted to Second Lieutenant in January, 2015. Having completed his military service, he currently continues his post in Ankara University as an Assistant Professor.

His research interests include different yet closely related aspects of computer science from image processing, computer vision and graphics to artificial intelligence and fuzzy logic as well as mathematical modelling and statistical analysis. He recently developed a vision-based user tracking system for various augmented reality applications for cultural heritage in particular. He setup the SAAT laboratory in the department for conducting research with the main aim for incorporating AI approaches for solving a wide range of real world problems.

Dr. Bostanci has been involved in technical committees for several conferences and acted as a reviewer for various journals.



Ali Emre Soylucicek is currently a senior student at Ankara University, Computer Engineering Department, Turkey. He graduated from Golbasi Anatolian High School. He completed Erasmus+ exchange program in Poland – Krakow for one year as well as participating in Erasmus+ internship program in Krakow. He performed his internship at a game company called "Duckie Deck". He is currently working part-time GelisimPark Inc. at Cyberpark-Bilkent.

He is focused on developing games as a programmer for both mobile and other platforms. He also conducted research about artificial intelligence implementations inside games. He improved himself by doing researches about Unity3D game engine. He created variety of games on both iOS and Android games, also published them to the market. He is currently developing more games and doing research in game development and artificial intelligence implementations inside games.